A TSP Software Maintenance Life Cycle

Chris A. Rickets
Naval Air Systems Command

Team Software Process℠ (TSP℠) and Personal Software Process℠ (PSP℠) have always been associated with software development, but what about TSP/PSP for software maintenance? This article discusses how TSP/PSP was adapted for use on a software maintenance project, resulting in a new proxy for estimating maintenance activity and the creation of a TSP software maintenance life cycle.

Anyone who has been exposed to Team Software Process℠ (TSP℠)/ Personal Software Process℠ (PSP℠) knows that its life-cycle model is based on software development, but what about TSP/PSP for software maintenance? This article tells how an inexperienced team created a TSP/PSP life-cycle model for corrective maintenance that allowed it to finish well ahead of schedule yielding a 76 percent increase in problems fixed and reducing defects by 38 percent.

Background

The Naval Air Systems Command's (NAVAIR) AV-8B Joint System Support Activity (JSSA) has successfully applied TSP/PSP to new software development and to software maintenance projects for three years.

In February 2001, AV-8B's Joint Mission Planning System (AVJMPS) team began applying the TSP/PSP traditional development life-cycle model to the AVJMPS software development project.

In the spring of 2002, AV-8B's Mission Support Computer (MSC) software engineering team began the H2.0 Block Upgrade maintenance software effort applying TSP/PSP. Their effort under H2.0 Block Upgrade was primarily maintenance. They found that by adapting the TSP life-cycle model to better fit maintenance activity, they were able to isolate and measure rework activities and drive down defect rates while increasing productivity.

Figure 1: *TSP Traditional Development Life Cycle Phases*

HLD	High-Level Design
HLDINSP	High-Level Design Inspection
DLD	Detailed-Level Design
DLDR	Detailed-Level Design Review
DLDINSP	Detailed-Level Design Inspection
CODE	Code
CR	Code Review
CODEINSP	Code Inspection
COMPILE	Compile
UT	Unit Test
IT	Integration Test
ST	System Test

Proxies were used to size each problem fix in terms of estimated hours instead of source lines of code (SLOC). There is typically no correlation between the solution of the problem and SLOC. There is a correlation between completion of the problem and time to fix.

The MSC Software Maintenance Team

The MSC Operational Flight Program (OFP), a real-time embedded program running on a PowerPC processor, provides mission computer functionality for the AV-8B Harrier II+. The OFP comprises 700,000 SLOC, mostly in C++.

The five-member H2.0 MSC software engineering team was relatively inexperienced. Two team members had experience working with the MSC OFP software and the toolset. One of these two was newly promoted to the software lead position having no prior experience in this position. Out of the three new team members, only one was familiar with the toolset, and none of them were experienced in the problem domain. The teams primary tasking was corrective maintenance (i.e., fixing software defects).

It was obvious to JSSA management and the H2.0 software team that the team required startup time to learn the toolset, the MSC OFP software, and TSP/PSP.

Maintenance and Development Life Cycles Are Not a Perfect Match

The MSC software engineering team began development with the traditional development life-cycle phases, shown in Figure 1, which are supported in TSP/PSP training and by the TSP/PSP tool. The team, however, soon realized that this life cycle did not address problems associated with software maintenance.

The team found that the TSP traditional life cycle, as shown in Figure 1, did not include a phase for problem identification. Identifying the problem, recreating the problem in controlled conditions, and identifying the solution are critical activities in the first steps of software maintenance. The ISO [International Organization for Standardization]/IEC [International Electronical Commission] Standard 12207 [1] describes these activities in the Problem Modification and Analysis step of the maintenance life cycle.

Next, the team noted that most software changes did not affect high-level design, and many did not affect detailed-level design. Most changes fell into the corrective maintenance category, resulting from missing or misinterpreted requirements, coding logic errors, or missing source code.

The team also experienced cases where finding the correct solution required iteration through the TSP life-cycle phases. For example, the proposed solution might have an unanticipated side effect, which required iteration back through design, code, and test. There are also cases where determining the complete scope of the problem required multiple probes into the design and code.

The traditional TSP life cycle does not accommodate iteration. While it is possible to add iterations through the phases to one's plan, the original work plan for development activity is based on a single iteration from high-level design through integration test. Consequently all the earned value is associated with the originally planned iteration, and additional iterations show up as unplanned work with no earned value.

The Lite Life-Cycle Model Is Born

TSP teaches that your process must be your own and that TSP can be adapted to fit the way your organization does business. The H2.0 MSC software engineering team adapted the TSP life cycle to fit their

team and the maintenance activity.

The H2.0 MSC software engineering team coined the phrases *classic* and *lite* to describe their software lifecycle models. The classic life-cycle model is the traditional TSP development model, while lite life-cycle model refers to the maintenance life-cycle model created by the team. The phases of the lite model, shown in Figure 2, specifically address the shortcomings noted in the previous section.

The lite life cycle adds phases and activities specific to software maintenance, and consolidates phases from the traditional life-cycle model to allow for iteration. The lite life cycle preserves review and inspection activities, although they are less visible and may be reordered from the traditional approach. The following paragraphs explain each of the lite life-cycle phases.

- **IDENT.** During the identification phase, the software engineer works with the systems engineer (SE)[1] to verify the requirements and demonstrate the problem. Early involvement of the SE ensures that the problem as specified in the Problem Report (PR) is correct. Often, the PR describes symptoms rather than identifying the root cause. The desired solution in the PR may also be incomplete. Given an understanding of the problem, the software engineer then identifies the cause of failure condition within the source code and demonstrates the source code problem and failure to a peer. If the peer agrees with the software engineer that the problem has been correctly and completely identified, the software engineer can then move to the INWRK [in-work] phase.
- **INWRK.** The in-work phase encapsulates design, code, and unit test to allow for iteration in the maintenance environment.
- **Design and Design Review.** If a design change is needed, the software engineer implements the changes and reviews the changes using his or her design review checklist. Inspection is deferred to the INSP [inspection] phase, except in cases where a substantial change to the design is required, or where there are special circumstances, e.g., a change to a class in the Common OFP[2]. The decision to delay inspection of small, simple design changes to the INSP phase is based on several factors. First, the cost of putting together an inspection package for a one- to three-line source code change outweighs the benefit. Second, the risk of delaying the inspection has proved to be acceptable. This risk is mitigated by the team's experience that small changes are typically identified in the IDENT phase, where a peer has already concurred with the change.
- **Code and Code Review.** Source code changes are made, based on baseline versions of the OFP. The software engineer reviews their changes using a code review checklist. Code inspection is deferred to the INSP phase.
- **Compile.** The source code is compiled until all code compiles cleanly.
- **Unit Test.** Unit testing is then performed. If the fix for the PR fails, the software engineer continues to solve the problem, iterating through design, code, and unit test until a successful solution is achieved and any negative side effects have been eliminated. When the desired results are obtained, the SE is shown the unit test results as an additional check that the solution is correct and complete. This conforms to the maintenance implementation ISO/IEC activity described in 5.5.3.2 (b) of Standard 12207 [1]
- **INSP.** The inspection phase consolidates both design and code inspections, except in cases where complex solutions require that design and code inspections be conducted separately. The inspection package includes the inspection log, modified design and source code files, the PR, a document describing where changes were made and why, and a copy of the unit test plan. The addition of the unit test plan accords with the maintenance implementation ISO/IEC activity as described in 5.5.3.2 (a) of Standard 12207 [1]. The addition of a document describing where and why changes were made is a road map for the inspector. The team made this a mandatory requirement when it was found that this information dramatically reduced inspection times.
- **IT.** Integration test is performed using a developmental baseline in the lab environment (i.e., the actual hardware). The software engineer uses the PR test plan, and the SE may perform additional tests to verify the correctness and completeness of the fix. If either the software engineer or the SE determines that the fix is incomplete, the software engineer continues to log his work in this phase. Once the software engineer and SE agree that the fix is correct, the PR is then considered completed and available for system testing; although, there are exceptions as indicated under the RA [rework assessment] and ST [system test] phases.
- **RA.** The rework assessment phase is added to a PR, in the TSP workbook, when questions arise about a completed PR that requires the software engineer to investigate and resolve. The software team found that there was often considerable time lag between completion of the IT phase and the closing of the PR by the SE. The SE would often have forgotten his or her initial consultation with the software engineer or would question whether the fix addressed something that may have been overlooked by the requirements. These questions would cause the software engineers to spend a considerable amount of time becoming reacquainted with a PR completed back under IT. This phase is used to capture time spent in determining if a software problem still exists for the PR in question. If no problem exists, the PR is closed by the SE. If a problem is identified, the ST phase is then entered.
- **ST.** When a problem is found in system test that is related to an allocated PR, an ST phase is added for that PR. The software engineer will log his or her time in this phase until the problem is resolved. Problems detected during the system test phase are used as the quality indicator for the H2.0 team. The team set its quality goal at having no more than 10% of PR's being rejected in system test.

IDENT	Identification
INWRK	In Work
INSP	Inspection
IT	Integration Test
RA	Rework Assessment
ST	System Test

Figure 2: *TSP Maintenance Life Cycle Phases*

A mapping of the development classic life-cycle phases to the maintenance lite life-cycle phases is shown in Figure 3 (see next page).

Customer's Perspective on TSP Maintenance Activities

The direct customer for the software team is the Block lead and the Integrated Product Team (IPT) lead. The customer defines project goals for maintenance projects and levels of success are established during TSP Meeting 1, just as in development projects. The customer participates in launches and postmortems,

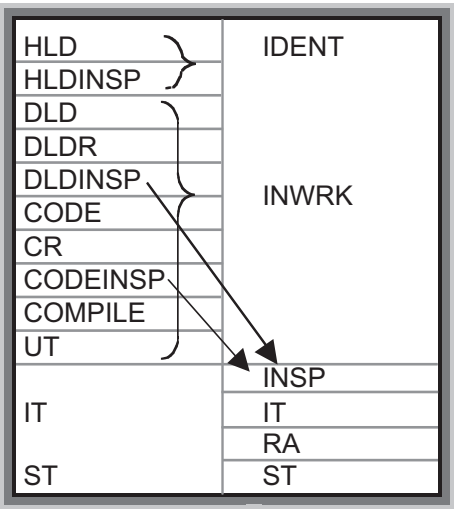

Figure 3: *Mapping from Development to Maintenance Life Cycle*

just as in development projects.

"TSP has brought credibility to estimates and commitments to perform worthwhile in a maintenance environment," said AV-8B JSSA's IPT Lead Dwayne Heinsma. "We are no longer questioned on the basis of estimating our requirements because we have the data and the performance to back it up."

Heinsma continued, "TSP has also contributed significantly to our ability to establish organizational improvement goals in the maintenance area. We did not have the historical measures that baselined our performance previously and today we now have a process by which we update our organizational performance baseline using project postmortem data."

Heinsma also stated, "Today, there is a strong push from NAVAIR leadership to establish improvement goals (productivity, quality, cost, and schedule) and show progress toward meeting those goals. With TSP, we have established the baseline; we have the improvement goals and the data showing progress toward the goals. If we are not meeting the goals, TSP provides us insight into what is holding us back, and we can focus on those elements that will help us improve most significantly."

The H4.0 Block Lead Greg Janson for the current software maintenance effort added, "I feel that TSP is worthwhile from a customer point of view. TSP provides a good quantitative tool for performance assessment. Qualitatively, it creates a solid basis for estimating that is difficult to question."

Results

The H2.0 team finished their tasking well ahead of schedule. In fact, the H2.0 team was able to reassess how many additional PR's could be solved in the time remaining on the project. The team initially estimated 102 PR's over a two-year period. The team actually completed over 180 PR's. This is a considerable accomplishment, given the team's relative lack of domain knowledge and the fact that the first two and one-half months of the project were spent completing TSP training. During this time, the H2.0 team also developed and documented processes for the H2.0 software development effort. The team came very close to their quality goal, achieving a 13 percent rejection rate in system test.

The AV-8B software team is now working on the next block upgrade to the AV-8B MSC OFP. They continue to refine the lite life cycle to improve software quality. In this block upgrade, their goal is to reduce the rework time measured in the RA phase, and to refine their defect logging to yield more fine-grained information earlier in the life cycle.◆

Reference

1. ISO [International Organization for Standardization]/IEC [International Electrotechnical Commission] 12207. ISO/IEC 12207: Information Technology – Software Life-Cycle Processes. 1st ed. ISO/IEC, 1995.

Notes

1. At the AV-8B JSSA, the systems engineering team is responsible for system and software requirements.
2. Common OFP is a basic set of software built upon for different aircraft with common fundamental requirements but differing missions and/or systems.

About the Author

Chris A. Rickets is a computer scientist in the software engineering group at the AV-8B Joint System Support Activity. He has been working on Embedded Avionics Systems for the past 14 years. He was the H2.0 Harrier Block Upgrade Software Lead and is currently working on the H4.0 Harrier Block Upgrade. Rickets has both a Bachelor of Science and Master of Science in computer science from California State University Chico.

CMDR, NAWCWD
41K300D MS 2004
507 E Corsair ST
China Lake, CA 93555-6110
Phone: (760) 939-5838
E-mail: chris.rickets@navy.mil

Web Sites

Software Process Dashboard Initiative

http://processdash.sourceforge.net

The Software Process Dashboard Project is an open-source initiative to create a Personal Software Process℠ (PSP℠)/Team Software Process℠ (TSP℠) support tool. The Process Dashboard is an existing support tool originally developed in 1998 by the U.S. Air Force, and has continued to evolve under the open-source model. It is freely available for download under the conditions of the Gnu's Not Unix Public License. The Process Dashboard supports data collection, planning, tracking, data analysis, and data export. The major strengths of the Process Dashboard are ease of use, flexibility/extensibility, platform independence, and price. The Team Process Dashboard is currently under development.

Software Engineering Institute

www.sei.cmu.edu

The Software Engineering Institute (SEI℠) is a federally funded research and development center sponsored by the Department of Defense to provide leadership in advancing the state of the practice of software engineering to improve the quality of systems that depend on software. SEI helps organizations and individuals improve their software engineering management practices. The site features complete information on models the SEI is currently involved in developing, expanding, or maintaining, including the Team Software Process℠, Personal Software Process℠, Capability Maturity Model® Integration, Capability Maturity Model® for Software, Software Acquisition Capability Maturity Model®, Systems Engineering Capability Maturity Model®, and more.